

NUME : COSTEA PAULA MARIA

GRUPA : 302210

PROFESOR LABORATOR : DAN MITREA

**ASSIGNMENT 4**

**FOOD DELIVERY MANAGEMENT SYSTEM**

**Requirement: Design and implement a food delivery management system for a catering company. The client can order products from the company’s menu. The system should have three types of users that log in using a username and a password: administrator, regular employee, and client.**

**1 . Obiective**

Principalul obiectiv al acestei teme este implementarea unei aplicatii care simuleaza un sistem de food delivery . Aceasta aplicatie are o parte care administreaza sistemul ( adauga produse in meniu , le editeaza sau le sterge ) si o parte care administreaza personalul ( employee , bucatar ) .

Ca obiective secundare avem :

* Utilizarea unui “ design patterns “ din Java ( “ Composite “ si “ Observer “ )
* Utilizarea unor colectii predefinite in implementarea restaurantului ( “ HashMap “ )
* Serializarea si deserializarea datelor

**2 . Analiza problemei , modelare , scenarii , cazuri de utilizare**

* **Analiza problemei :**

Principalul scop este de a dezvolta un program ce implementeaza toate cerintele cerute , si , in acelasi timp , foloseste paradigmenele Programarii Orientate pe Obiect . Pentru a face asta , avem nevoie sa gasim un mod sa ne stocam informatiile , si pentru asta , alegem “ HashSet “ pentru a retine meniului restaurantului , o lista cu toate comenzile effectuate , si un “ HashMap “ , care are ca si cheie comanda , si ca valoare , o lista de “ MenuItems “ . Oricum , adevarata provocare este sa implementam un anumit “ desing pattern “ , si sa serializem si sa deserializam obiectele noastre , pentru a nu pierde datele cand aplicatia este inchisa .

* **Modelarea :**

1. **Composite Design Pattern** este un “ design pattern “ de partitionare si descrie un grup de obiecare ce e tratat ca o singura instanta a aceluias tip de obiect . Intentia sa este sa “ compuna “ obiectele in 3 structuri , ce reprezinta ierarhii partiale . Acest lucru ti permite sa ai o structura de tip “ arbore “ , si sa faci fiecare nod din structura arborelui sa efectuezre un anumit task .

Composite Design Patern are 4 participanti :

1. Componenta : Componenta declara interfata pentru obiectele din compozitie si pentru accesul si controlul componentelor-copii . Implementeaza , de asemenea , in mod implicit , si un comportament pentru interfata comuna tuturor claselor , dupa caz .
2. Frunzaele : Frunzele definesc comportamentul pentru obiectele primitive din compozitite .
3. Compozitia : Compozitia stocheaza componentele-copii si implementeaza operatii legate de copii , in interfata de componente .
4. Clientul : Clientul manipuleaza compozitia obiectului prin interfata componentelor .
5. **Observer Pattern** este utilizat cand avem relatii de tip “ one – to – many “ intre obiecte , astfel incat daca un obiect este modificat , obiectele sale dependente sunt informate automat . “ Observer Pattern “ se incardreaza in categoria modelului comportamental .

“Observer Pattern “ utilizeaza 3 clase :” Subject “ , ” Observer “ si “ Client “ . “ Subject “ este un obiect cu metode ce ataseaza si detaseaza observatorii de obiectele “ client “ .

Pentru a ne administra datele in mod eficient , utilizam :

1 . **HashSet :** “ HashSet “ implementeaza intergata “ Set “ , sprijinita de un “ hash table “ , care este , de fapt , o instant “ HashMap “ . Nu ofera nicio garantie cu privire la ordinea de interatie a setului , ceea ce inseamna ca , clasa nu garanteaza ordinea constanta a elementelor de-a lungul timpului . Aceasta clasa permite elemenul “ null “ . Clasa poate oferi , de asemenea , performanta timpului constant , pentru operatii de baza , ca “ add ” , “ remove “ , “ contains “ and “ size “ , presupunand ca functia “ hash “ disperseaza elementele corespunzator .

2. **HashMap** : “ HashMap “ asigura implementarile de baza ale intefetei “ Map “ . Stocheaza informatiile in perechi ( Key , Value ) . Pentru a accesa o valoare , trebuie sa ii cunoastem cheia . “ HashMap “ este cunoscuta ca “ HashMap “ , deoarece utilizeaza o tehnica numita “ Hashing “ . “ Hashing “ este o tehnica de convertire a unui String mare , intr-un string mic , ce reprezinta de fapt acelasi String . O valoare mai mica ajuta in indexarea si cautarea mai rapida . “ HashSet “ utilizeaza intern “ HashMap “ . Intern , foloseste un “ link list “ pentru a stoca perechi “ key - value “ .

Pentru a ne pastra datele dupa ce aplicatia se inchide , folosim “ serializarea “ :

Java ofera un mecanism , numit “ object serialization “ , unde un obiect poate fi reprezentat ca o secventa de biti , ce include informatiile obiectului , precum si informatii despre tipul obiectului si tipurile de date stocate in obiect .

Dupa ce un obiect serializat a fost scris intr-un fisier , el poate fi citit din fisier si deserializat , ceea ce reprezinta , de fapt , tipul informatiei , si bitii ce reprezinta obiectul si informatiile lui pot fi utilizate pentru a crea obiectul in memorie .

* **Scenarii**

Pentru ca aplicatia sa ruleze asa cum ne dorim , datele de intrare trebuie introdusa intr-o maniera potrivita . Cand vrem sa introducem un produs compus , dupa ce ii scriem numele , trebuie sa adaugam la rubric “ componente “ ,doar produse ce se afla deja in meniu ; de asemenea ,aceste produse trebuie despartite prin virgule! .

* **Cazuri de utilizare**

La inceputul programului , utilizatorul va vedea o serie de butoane ,si trebuie sa isi aleaga rolul in interactiunea pe care o are cu aplicatia ( administrator , client sau employee ) .Fereastra administratorului permite crearea ,stergerea si editarea produselor din meniul restaurantului . Daca administratorul doreste sa introduca un produs de baza in meniu , o sa adauge numele si pretul acestui produs in panel-ul de sub eticheta “Add Base Product “,dupa care va apasa pe butonul “Add to menu “ . In cazul in care acest produs exista deja in meniu , va aparea un mesaj . In cazul in care vrem sa adaugam un produs compus , o sa ii trecem numele si produsele pe care le contine ,care trebuie sa se afle deja in meniu , si trebuie sa fie despartite prin virgule , in panel-ul de sub eticheta “ Add Composite Products “ . In caz ca numele dat exista deja ,sau unul din produsele componente nu exista in meniu ,sau nu sunt despartite prin virgule ,va aparea un mesaj de eroare . Pentru a sterge un produs ,trecem in panel-ul de sub eticheta “ Delete Product “ numele ,si apasam butonul “ Delete” ; daca cumva se va sterege un produs care e inclus in alt produs compus , se va sterge si acesta . Daca dorim sa editam un produs ,ii trecem numele vechi ,numele nou si pretul in panel-ul de sub eticheta “ Edit Product “. Fereastra clientului permite crearea unei noi comenzi, afisarea pretului unei comenzi , si generarea bonului pentru un anumit ID . Pentru a adauga o noua comanda ,trebuie sa trecem, separate prin virgule , produsele din meniu. Fereastra angajatului se deschide cand clientul vrea sa introduca o comanda, angajatul trebuind sa apese pe “ok “ ,daca doreste sa preia comanda , sau “cancel “ ,daca va prelua mai tarziu comanda

**3 . Proiectarea**

Pentru a respecta paradigmele Programarii Orientate pe Obiect , am impartit programul in mai multe pachete ce contin mai multe clase .

Primul pachet pe care il implementam este “ DataLayer “ , pachet ce contine clasele “ Serializator “ si “FileWriterr “ . In clasa “ Serializator “ am implementat 2 metoda : o metoda pentru serializare , si cealalta pentru deserializarea obiectului restaurant . In clasa “ FileWriterr “ avem un constructor , si o metoda , folosita pentru a creea un nou fisier .txt , folosita in generarea bonului unei comenzi .

Al doilea pachet , numit “ BusinessLayer “ contine :

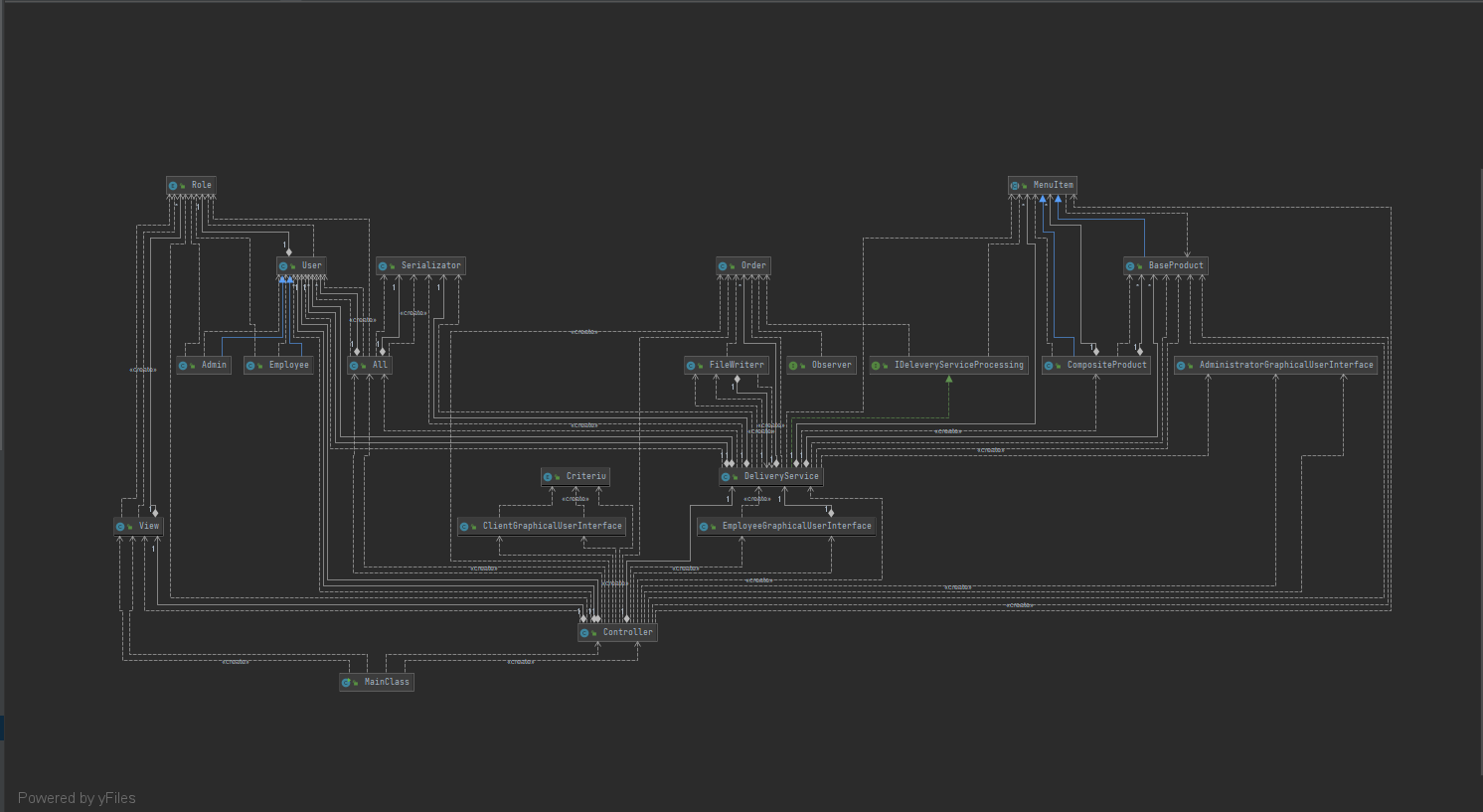
* Clasa “ BaseProduct “ , care contine 2 atribute : “ price “ and “ name “ ,si getters si setters pentru fiecare atribut .
* Clasa “ CompositeProduct “ , care contine aceleasi attribute ca si clasa “ BaseProduct “ , si , in plus , contine si o lista cu produsele de baza ale unui produs compus ; dar si getters si settes pentru fiecare atribut.
* Clasa “ BaseProduct “ si “ CompositeProduct “ extend clasa abstracta “ MenuItems “ . Aici definim cateva metoda abstracte pe care urmeaza sa le implementam in ambele clase care extend clasa “ MenuItems “ . Aceasta clasa implementeaza interfata “ Serializable “ .
* Mai apoi , implementam clasa “ Order “ ,care are ca attribute “ orderId “ , ”date “, si getters si setters pentru fiecare atribut ; aceasta clasa implementeaza clasa “ Serializable “
* Interfata “ IDeliveryServiceProcessing “ defineste cateva metode pentru admin si pentru ospatar
* Toate aceste metode vor fi implementate in clasa “ DeliveryService “ . Atributele aceste clase sunt : “ ordersList “ ( comenzile ,impreuna cu produsele din meniu pe care le-a comandat ) , “ orders “ (lista comenzilor ) , “ menu “ (meniul). Unele dintre cele mai importante metode implementate in aceasta clasa sunt acele metode pentru crearea , stergerea , editarea produselor din meniu , si acelea pentru crearea unei comenzi , calcularea pretului pentru o comanda , si generarea bonului pentur o anumita comanda .

In continuare , voi prezenta pachetul “ PresentationLayer “ ,care contine clasele GUI .

GUI este o forma de interfata grafica , care permite utilizatorului sa interactioneze cu dispozitivile electronice , prin imaginile grafice si indicatori vizuali .

Avem o clasa petru “ employee “ , o clasa pentru “ admin “ , si o clasa pentru “ client“ . Pe langa acesteaza , avem o pagina de start , si un “ Controller “ ( unul principal , si unul pentru admin si unul pentru waiter ) - aici sta de fapt logica claselor GUI . In “ Controller “ , avem “ Action Listeners “ , metoda pentru crearea produselor din meniu si alte elementru GUI . GUI utilizeaza ferestre , imagini , tabele , etichete , pentru a executa comenzi .

Diagrama UML :



**4.Implementarea**

* Clasa “ Serializator “ are 2 metode :
* Serialization ( Object o ) : ia datele din parametrul “ DeliveryService “ si le pune intr-un fisier
* Deserialization ( ) : creeaza o noua instanta a restaurantului folosind informatiile salvate in fisier in urma serializarii .
* Clasele : “ BaseProduct “ & “ CompositeProduct “ au doar metodele getters si setters si metoda pentru calcularea pretului definite in clasa abstracta “ MenuItem “
* Clasa “ Order “ : contine getters si setters pentru fiecare atribut , si , in plus , avem si clasa hashCode ,care calculeaza codul functiei Hash al fiecarei comenzi .
* Clasa “ DeliveryService “ are mai multe metode :
* Metoda “ createNewMenuItem() : Metoda care adauga la meniu un alt produs(un alt obiect de tip "menuItem " )
* Metoda “ findBaseProduct ( ) : Metoda care gaseste in meniu un produs "menuItem",dat ca parametru .Aceasta metoda cauta produsul "menuItem",si printre componentele unui produs compus Daca il gaseste, returneaza produsul,iar in caz contrar,returneaza null.
* Metoda “ findMenuItem ( ) : Gaseste un produs in meniu .
* Metoda “deleteItemMenu “ : Sterge din meniu produsul "menuItem",dat ca parametru
* Metoda “ editMenuItem “ : Metoda care inlocuieste un produs al meniului cu unul nou
* Metoda “ findOrder “ : Metoda care verifica daca exista o comanda la o anumita masa
* Metoda “noItems “ : Metoda care verifica daca avem o comanda goala(fara produse)
* Metoda “createNewOrder “ : Metoda care creeaza o noua comanda
* Metoda “ computePrice “ : Metoda care returneaza suma totala a unei comenzi
* Metoda “generateBill “ : Metoda care genereaza factura unei comenzi
* Clasa “ controller “ constituie logica programului .
* Clasa “EmployeeGraphicalUserInterface “ , nu este controlata de controller ; fereastra angajatului se deschide in momentul in care un client doreste sa introduca o comanda , angajatul putand prelua comanda ,apasand “ok “ ,sau putand amana comanda ,pentru mai tarziu , apasand “cancel “.

**5.Rezultate**

Rezultatele in urma rularii acestui program sunt stocate in fisierele .ser , iar ,datorita deserializarii ,aceste operatii sunt vizibile si dupa ce programul este inchis,putandu-se vedea in ferestrele administratorului , si a clientului , tabele cu meniul si cu comenzile efectuate .

Se poate observa in folderul incarcat, rezultatele obtinute in urma generarii unor bonuri , sub forma unor fisiere text .

**6.Concluzii**

In concluzie , aceasta tema ne-a introdus in lumea “ Design-Pattern-ului “ din Java ( Composite si Observer ,in mod special ) ; si , de asemenea , ne introduce conceptul de serializare .

**7. Bibliografie**

<http://www.tutorialspoint.com/java/java_serialization.htm>

<https://www.baeldung.com/java-serialization>

<https://www.geeksforgeeks.org/serialization-in-java/>

<https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html> <http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html>

<http://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html>

<http://javarevisited.blogspot.ro/2012/01/what-is-assertion-in-java-java.html>

<http://stackoverflow.com/questions/11415160/how-to-enable-the-java-keywordassert-in-eclipse-program-wise>

<https://intellij-support.jetbrains.com/hc/en-us/community/posts/207014815-How-toenable-assert>

<https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#tag>